บุญเลิศ เอี่ยมทัศนา

SOFT MAIL

ซอฟต์แวร์ชิพ: สกุล Collection
คลังสะสมข้อมูล

ไม่ว่าจะมีข้อมูลมากแค่ไหน ไม่ว่าข้อมูลนั้นจะเป็นแบบใด นึกถึงโอโอพีแล้วจะรู้ว่า Collection ช่วยได้เสมอ

ผู้เขียนต้องขออภัยท่านผู้อ่านเป็นอย่างมากที่บทความชุดนี้ขาดตอนไปหลายฉบับเพราะห่างเครื่อง แต่ยังดีที่มีหนังสือเกี่ยวกับการเขียนโปรแกรมในแบบโอโอพีกับ C++ ติดไปด้วย จึงมีเกล็ดเกี่ยวกับโอโอพีมาเล่าสู่กันฟัง บทความตอนนี้ผู้เขียนตั้งใจจะสร้างชิพจากอีกสกุลหนึ่งต่อเนื่องจากของเดิม เพื่อทำให้เป็นโอโอพียิ่งขึ้น

สำคัญที่ความเป็นเวอร์ชวล

ผู้เขียนลองอ่านทวนบทความของตัวเองเกี่ยวกับการเขียนโปรแกรมในแบบโอโอพีที่ลงในไมโครฯ ฉบับที่ 59 และ 60 ผู้เขียนได้กล่าวถึงความเป็นเวอร์ชวลของสกุลไว้ไม่ชัดเจน จึงขอชี้แจงตรงนี้ก็แล้วกัน

- ความเป็นเวอร์ชวลไม่ได้เป็นเพราะมีกิจกรรมเป็นเวอร์ชวลแต่เพียงอย่างเดียว

- ความจำเป็นที่ต้องกำหนดให้สกุลหนึ่งสกุลใดเป็นเวอร์ชวลนั้นเกิดจาก หนึ่งความต้องการให้กำหนดนิมิตเป็นแบบไดนามิกส์ได้ และ/หรือสองมีกิจกรรมเป็นเวอร์ชวล

- การกำหนดสกุลให้เป็นเวอร์ชวลทำได้โดยเลือกกิจกรรมที่ต้องปฏิบัติก่อนกิจกรรมอื่น (เช่น Init) มากำหนดเป็น constructor และให้กิจกรรมที่ยกเลิกนิมิต (instance : ตัวแปรของสกุล) เป็น destructor

คุยกันเรื่องโอโอพี

หลังจากอ่านหนังสือหลายเล่มต่างก็กล่าวถึงการเขียนโปรแกรมในแบบโอโอพีกันต่าง ๆ นานา ผู้เขียนพบว่า

ความมุ่งหมาย

จุดมุ่งหมายของการเขียนโปรแกรมในแบบโอโอพีก็เพื่อทำให้โปรแกรมเมอร์สามารถพัฒนาโปรแกรม (ตั้งแต่เริ่มออกแบบจนกว่าโปรแกรมจะใช้งานได้ดี) ได้ในเวลาอันสั้นที่สุด โดยไม่ค่อยคำนึงถึงขนาดของโปรแกรมและเวลาที่ใช้รันซึ่งจะทำให้

- สามารถนำโปรแกรมเก่ามาใช้ใหม่ได้ และเมื่อต้องการแก้ไขปรับปรุงก็ดำเนินการได้โดยไม่ต้องยุ่งเกี่ยวกับซอร์สโค้ดเดิม

- แยกกันเขียนโปรแกรมได้อย่างเป็นอิสระต่อกัน

ลักษณะของภาษาโอโอพี

จากความมุ่งหมายทำให้เกิดความต้องการภาษาที่สามารถเขียนโปรแกรมให้

- เป็นเอกเทศ (Encapsulation) คือสามารถเก็บข้อมูลและโปรแกรมที่จะดำเนินการกับข้อมูลนี้ซึ่งเรียกว่ากิจกรรม (method) ไว้ในที่แห่งเดียวกัน

และอยู่ในที่ปกป้องอันหนึ่ง (เข้าถึงไม่ได้) เรียกว่าเป็นการกำหนดสกุล (class)

- สืบสกุล (Inheritance) คือสามารถสืบสกุล เพื่อแก้ไขปรับปรุงโปรแกรมในบรรพบุรุษได้โดยไม่ต้องแตะซอร์สโค้ดเดิม

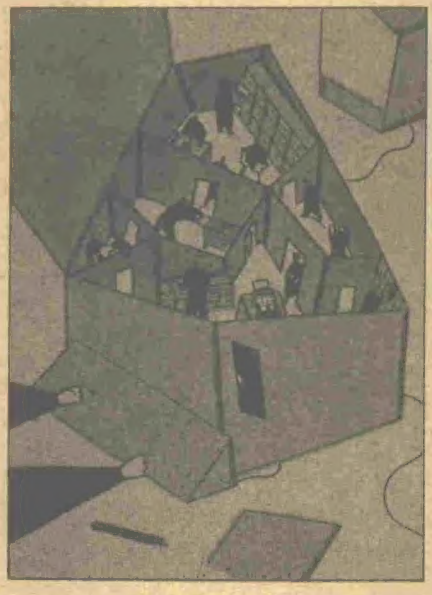
- หลายรูปแบบ (Polymophism) เป็นการขยายการสืบสกุลให้สามารถใช้สื่อความ (ชื่อกิจกรรม) เดิม แต่การปฏิบัติใหม่แตกต่างไปจากเดิมได้

- โอเวอร์โหลด (Overloading) เป็นการขยายการสืบสกุลเหมือนกัน คือสามารถกำหนดชื่อกิจกรรมซ้ำซ้อนได้ เช่น เขียนกิจกรรม Power (a,b) เป็นฟังก์ชันเพื่อให้ค่า a ยกกำลัง b ไว้เป็นเลขจำนวนเต็ม เมื่อต้องการเขียนใหม่ให้เป็นเลขจำนวนจริงก็สามารถดำเนินการได้โดยใช้คำว่า Power แต่ผ่านค่าเป็นเลขจำนวนจริง ซึ่งความสามารถของภาษาจะต้องตรวจได้ว่าค่าที่ผ่านมาเป็นเลขจำนวนเต็มหรือจำนวนจริง และเรียกใช้โปรแกรมย่อย Power ที่ตรงกับแบบข้อมูล

ถือกันว่าภาษาที่จะเป็นโอโอแอล (OOL:Object Oriented Language) ได้คือภาษาที่มีคุณสมบัติทั้ง 4 ข้อนี้ครบถ้วน ซึ่งมีอยู่ในภาษาสมอลล์ทอล์คและ C++ สำหรับเทอร์โบปาสคาล 5.5 ขาดเรื่องการโอเวอร์โหลด ซึ่งผู้เขียนเห็นว่ามีที่ใช้น้อยจึงไม่น่าจะตัดออกมาจากความเป็นโอโอแอล

โอโอพีกับโอโอพี

จากหนังสือที่กล่าวถึงการเขียนโปรแกรมแบบโอโอพี ผู้เขียนพอจับประเด็นได้อีกประเด็นหนึ่งในเรื่องของความเข้มงวด ซึ่งจะตรงข้ามกับขนาดของโปรแกรมและความเร็วในการรัน คือยิ่งเข้มงวดมากขนาดของโปรแกรมจะขยายใหญ่ ตัวโปรแกรมจะรันได้ช้า การที่ตั้งประเด็นนี้ขึ้นก็เพราะ C++ อาจกล่าวได้ว่าเป็นโอโอแอลภาษาที่สองต่อจากสมอลล์ทอล์ค จึงให้ผู้กำหนดสกุลเลือกได้ว่าจะเข้มงวดมากน้อยเท่าไร เรื่องสำคัญอยู่ที่การเข้าถึงข้อมูลและกิจกรรม ความหมายของคำว่าเข้มงวดตามภาษาสมอลล์ทอล์คสำหรับข้อมูลแล้วจะไม่มีผู้ใดเข้าถึงได้เลย ต้องจัดกิจกรรมไว้ให้โดยเฉพาะ ทำให้ขนาดของโปรแกรมโตขึ้นและรันได้ช้าลง ส่วนกิจกรรมให้เข้าถึงได้ตามที่กำหนด แต่ถ้าไม่เข้มงวดเช่นในหนังสือเกี่ยวกับ C++ บางเล่มและโดยเฉพาะของดร.ยาร์น สเทราสทรูป (Bjane Stroustrup) ผู้พัฒนา C++ เองยินยอมให้สกุลถูกเข้าถึงข้อมูลได้โดยตรง ส่วนกิจกรรมจะเข้าได้ตามที่กำหนดไว้



ส่วนเทอร์โบปาสคาล 5.5 ตามหลักของภาษาสกุลจะเทียบเท่ากับเรคอร์ดเท่านั้น คือใคร ๆ ก็เข้าถึงข้อมูลและกิจกรรมได้หมด (คาดว่าคงมีการแก้ไขในรุ่นต่อไปให้เหมือนกับ C++) จึงขึ้นอยู่กับผู้เขียนโปรแกรมจะยึดถือหลักการเข้มงวดมากน้อยเพียงไร ที่ผ่านมาผู้เขียนถือหลักเข้มงวดแบบสมอลล์ทอล์ค แต่เมื่อมาพิจารณาตาม C++ แล้วเห็นว่าไม่มีความจำเป็นใดที่จะต้องเข้มงวดขนาดนั้น ขอให้ผู้ใช้โปรแกรมดำเนินการตามขั้นตอนนี้คือ

- ให้เฉพาะผู้สืบสกุลเท่านั้นที่เข้าถึงข้อมูลได้

- สำหรับโปรแกรมย่อยที่ใช้เป็นการภายในของสกุลผู้เขียนไม่ถือว่าเป็นกิจกรรมและจะกำหนดไว้หลังหมายเหตุว่า "Internal subroutine" กรุณาอย่านำไปใช้ในสกุลรองหรือใช้เป็นสื่อความ

การเขียนโปรแกรมแบบโอโอพี

ผู้เขียนควรจะใช้คำว่าการออกแบบแต่ฟังดูยิ่งใหญ่เกินไปสำหรับหลักการเขียนโปรแกรมแบบโอโอพี เนื่องจากเป็นเรื่องใหม่ ผู้เขียนจึงพยายามหาจากหนังสือที่เขียนไว้กระจัดกระจายในที่ต่าง ๆ เท่าที่จะหาได้ (ตามที่แสดงไว้ท้ายบทความนี้) จะมีก็แต่หนังสือของดร.ยาร์นที่เขียนไว้อย่างชัดเจน (แต่น่าเสียดายที่ไม่มีตัวอย่างประกอบ) จึงนำมาถ่ายทอดสู่ผู้อ่าน แต่ไม่ได้ยกมาโดยตรงทีเดียว ซึ่งเรื่องนี้ทั้ง ดร.ยาร์นและตัวผู้เขียนเองขอเรียนให้ทราบว่าจะถือเป็นหลักไม่ได้ เพียงแค่พอให้เห็นแนวทางเป็นใช้ได้

ขั้นตอนการออกแบบ

จะให้กระจายงานออกเป็นงานรองและรองตามลำดับ แล้วพิจารณาว่า

1. งานรองใดที่เป็นเอกเทศสมบูรณ์ในตัวเองให้กำหนดเป็นสกุล เช่น จะเขียนโปรแกรมคำนวณจำนวนค่าโทรศัพท์ทางไกลอาจแบ่งงานได้เป็น

- งานจับเวลา
- งานคำนวณค่าบริการ

จะเห็นว่างานทั้งสองชนิดเป็นเอกเทศและสมบูรณ์ในตัวเองนำไปกำหนดเป็นสกุลได้ และจะมองในแบบนามธรรมคือเป็นนาฬิกาจับเวลา เป็นต้น (สำหรับนาฬิกาจับเวลาสามารถนำไปใช้ในโครงงานอื่นได้อีกด้วย)

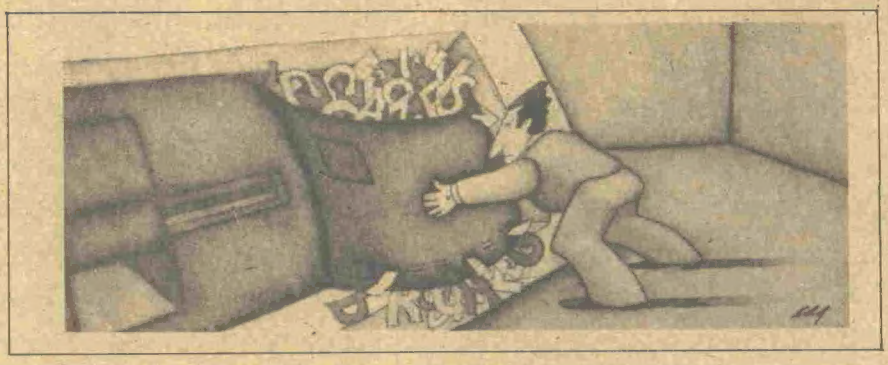
2. หากมีสิ่งหนึ่งสิ่งใดที่มองเห็นเป็นตัวตนถือเป็นรูปธรรม ให้กำหนดเป็นออปเจค (นิมิต) ของสกุล เช่น จะจับเวลาโทรศัพท์หลาย ๆ เครื่อง จะเห็นว่าแต่ละเครื่องต้องการนาฬิกาจับเวลา 1 เรือน คือนิมิตหนึ่งในหลายนิมิตนั่นเอง

3. ถ้าหลาย ๆ สกุลมีบางสิ่งบางอย่างเหมือนกัน เช่น ในงานบัญชีขายกับบัญชีพนักงานจะมีงานเกี่ยวกับชื่อและที่อยู่เหมือนกัน ให้ทำเป็นต้นสกุล (ทั้งนี้เพื่อไม่ให้มีงานที่ซ้ำซ้อนกัน และการเขียนโปรแกรมในแบบโอโอพีไม่ให้ใช้โปรแกรมย่อยภายนอก)

วิธีการเขียนโปรแกรม

1. ไม่ให้ใช้ข้อมูล (ตัวแปร) ที่เป็นโคบอล
2. ห้ามใช้ข้อมูลในสกุลอื่นโดยตรง
3. ห้ามใช้โปรแกรมย่อยที่กำหนดไว้นอกสกุล
4. ให้กำหนดกิจกรรมเท่าที่จำเป็น ห้ามกำหนดไว้ล่วงหน้า (เพราะสามารถเพิ่มเติมได้เมื่อต้องการ)

ผู้เขียนจะพยายามดำเนินการตามวิธีนี้ แต่กฎทุกกฎมีข้อยกเว้นด้วยเหตุผลที่จะแจ้งให้ทราบ



เข้าเรื่องสกุล Collection

สกุล Collection เป็นสกุลหนึ่งในภาษาสมอลล์ทอล์ค ทำหน้าที่เก็บข้อมูลทุกแบบไว้ด้วยกันไม่จำกัดจำนวน (จนกว่าจะเต็มหน่วยความจำของเครื่อง) และเป็นต้นสกุลของสกุลอื่นต่อไป เช่น IndexCollection ซึ่งเคยกล่าวถึงในหนังสือที่เกี่ยวกับโอโอพีด้วย C++

การที่ผู้เขียนหยิบยกสกุลนี้ขึ้นมาอรรถาธิบายนั้นมีจุดประสงค์เพื่อให้เป็นตุ๊กตาสำหรับแสดงการเขียนโปรแกรมในแบบโอโอพีเฉพาะขั้นตอนของการเขียน ดังนั้นจึงไม่ได้กล่าวถึงที่มา สำหรับทางด้านการทำงานจะใช้เก็บข้อความที่ผู้ใช้ป้อนในโปรแกรมเอดิเตอร์ ซึ่งจะนำไปเชื่อมต่อกับสกุล WindowMove เพื่อเป็นโปรแกรมเอดิเตอร์ที่สมบูรณ์ต่อไป จากนี้ไปผู้เขียนจะอธิบายถึงการกำหนดสกุล Collection อย่างค่อนข้างละเอียด

ระบุสื่อความ

จุดเริ่มต้นของการเขียนโปรแกรมแบบโอโอพีในขั้นตอนของการสร้างหรือกำหนดสกุลคือ ทบทวนวัตถุประสงค์หรือความมุ่งหมายในการกำหนดสกุลนี้ว่าเพื่ออะไร ซึ่งจะได้คำตอบว่า สำหรับเก็บข้อมูลทุกแบบไม่จำกัดจำนวน

ขั้นตอนต่อไปจะมองสกุลนี้ในฐานะออปเจคคือที่เก็บข้อมูล ซึ่งเราต้องการสื่อความใดบ้างที่จะติดต่อกับสกุลนี้ ผู้เขียนได้ตั้งชื่อเป็นภาษาอังกฤษไว้ใช้ในโปรแกรมอันได้แก่

- Init เพื่อเตรียมค่าต่าง ๆ เมื่อเริ่มต้น (มีในทุกสกุล)
- AddItem เพื่อเก็บข้อมูล 1 ข้อมูล (ซึ่งอาจจะเป็นเรคอร์ด สตริง เลขจำนวน หรือแม้แต่อักขระเพียง 1 อักขระ)
- DelItem เพื่อถอนข้อมูล 1 ข้อมูล
- GetItem เพื่ออ่านข้อมูล 1 ข้อมูล
- Fin เพื่อการยกเลิกนิมิต

เมื่อมองในแง่ของโอโอพีจะเห็นว่าสื่อความที่ใช้กับที่เก็บข้อมูลมีเพียงเท่านี้ แต่หากมองในแง่ของการดำเนินกรรมวิธีข้อมูลโดยทั่วไปยังขาดเรื่องเหล่านี้คือ

- ReplaceItem เพื่อนำข้อมูลใหม่ไปแทนที่ข้อมูลเก่า 1 ข้อมูล
- InsertItem เพื่อแทรกข้อมูล 1 ข้อมูล

จุดนี้เองหากเราเพิ่มความเข้มงวดคือไม่มองไปข้างหน้าก็จะไม่มีเรื่องหรือสองเรื่องนี้ แต่ผู้เขียนเห็นว่าน่าจะมีเพื่อให้สมบูรณ์จึงกำหนดเอาไว้

ระบุตัวแปร

เมื่อเราทราบสื่อความซึ่งในขั้นตอนของการกำหนดสกุลเรียกว่ากิจกรรม เราก็สามารถที่จะระบุตัวแปรตัวหนึ่งหรือหลายตัวเพื่อรองรับ (เก็บ) ออปเจค เช่น ออปเจคเป็นวงกลมก็จะต้องกำหนดตัวแปร 2 ตัวสำหรับเก็บโคออร์ดิเนตของจุดศูนย์กลาง อีกตัวแปรหนึ่งเก็บรัศมี

ตามตัวอย่างนี้ออปเจคคือที่เก็บข้อมูลได้ทุกชนิดไม่จำกัดจำนวน ซึ่งจะตีความได้ดังนี้คือ

- ที่เก็บข้อมูล ในการใช้งานโดยทั่วไปไม่ได้กำหนดให้เป็นเฉพาะที่เก็บข้อมูล อาจเป็นที่พักข้อมูลสำหรับรอดำเนินการ เช่น นำมาเข้าคิวไว้ หรือพักไว้เพื่อรอการเรียงลำดับข้อมูล และข้อมูลตัวจริงจะเก็บไว้ในแบบสแตติกหรือไดนามิกก็ได้ (คือเป็นตัวแปรหรือเก็บในฮีพ) ดังนั้นแบบการเก็บที่เหมาะสมคือเก็บเป็นพอยน์เตอร์ไปยังข้อมูล

- ได้ทุกชนิด พอยน์เตอร์ที่จะชี้ไปยังข้อมูลทุกแบบได้คือแบบ Pointer เพราะมีสื่อความที่จะถอนข้อมูลซึ่งอาจต้องยกเลิกข้อมูลในฮีพด้วย ดังนั้นจะต้องมีตัวแปรสำหรับเก็บขนาดของข้อมูลเป็นไบต์ด้วย แบบการเก็บที่เหมาะสมสำหรับข้อมูล 1 ข้อมูลจึงเป็นเรคอร์ดของขนาดและ Pointer ดังที่กำหนดให้แก่ OneData ในรูปที่ 1 และถือว่าเป็นข้อมูล 1 ข้อมูลในการกล่าวถึงต่อไป

- ไม่จำกัดจำนวน แบบการเก็บที่เหมาะสมคือลิงก์ลิสต์ (ซึ่งแตกย่อยไปได้อีกหลายแบบ) แต่ผู้เขียนมองถึงการใช้งานขั้นต่อไปที่ต้องใช้อินเด็กซ์ในการเข้าถึงข้อมูล จึงต้องทำให้เป็นอะเรย์ สำหรับการไม่จำกัดจำนวนจะทำได้โดยให้อะเรย์นี้อยู่ในฮีพ ดังนั้นแบบข้อมูลสำหรับรองรับออปเจคจึงเป็นพอยน์เตอร์ไปยังอะเรย์ของเรคอร์ดดังที่กำหนดให้แก่ buff ในรูปที่ 1

```
UNIT  COLLECT;

Interface

TYPE

   OneData  = RECORD
        size  : Word;
        dptr  : Pointer;
              END;
   BufferT  = ARRAY[0..10] OF OneData;
   BuffPtrT = ^BufferT;

   Collection = OBJECT
        buff     : BuffPtrT;
        stndNum  : Word;
        buffSize : Word;
        lastPstn : Word;
        errnum   : Word;
        break,extra : Boolean;

     CONSTRUCTOR  Init(numOfItem : Word);
     function     AddItem(len : Word; newitem : Pointer) : Word;
     function     DelItem(pstn : Word) : Boolean;
     function     ReplaceItem(pstn,len : Word; newitem : Pointer) : Boolean;
     function     InsertItem(pstn,len : Word; newitem : Pointer) : Boolean;
     function     GetItem(pstn : Word) : Pointer;
```

รูปที่ 1

เรื่องที่ขอชี้แจงเพิ่มเติมคือ ตามแนวความคิดของผู้เขียนจะให้ผู้เขียนโปรแกรมคนต่อไปเป็นผู้กำหนดขนาดคือจำนวนช่องของอะเรย์ จากจำนวนช่องของอะเรย์ เช่น 100 จะนำมาคูณ 6 นำไปจองหน่วยความจำในฮีพสำหรับอะเรย์นี้ ในการใช้งานเมื่อข้อมูลเต็มก็จะจองหน่วยความจำในฮีพใหม่เป็น 125 (เพิ่มครั้งละ 25%) ลอกข้อมูลในอะเรย์เดิมไปอะเรย์ใหม่แล้วยกเลิกอะเรย์เดิม ด้วยวิธีเช่นนี้ทำให้สามารถขยายที่เก็บข้อมูลคือ buff ได้จนกว่าจะเต็มหน่วยความจำ

ตามรูปที่ 1 กำหนดให้เป็นอะเรย์ 0..10 หรือกำหนดให้เป็น 0..1 ก็ได้ เป็นการบ่งว่าทับช่องแรกเป็นศูนย์ และสำหรับขนาดของข้อมูลในเรคอร์ดผู้เขียนได้ออกแบบไว้ว่า ถ้าข้อมูลจริงเป็นสแตติกหรือไดนามิกที่ไม่ต้องการยกเลิกจะให้ค่าเป็นศูนย์

ตัวแปรประกอบ

นอกจากตัวแปรของออปเจคแล้ว ส่วนมากจะต้องกำหนดตัวแปรขึ้นอีกจำนวนหนึ่งไว้ช่วยเหลือการทำงานของกิจกรรม ซึ่งในการเขียนโปรแกรมจริงนั้นจะเกิดขึ้นเมื่อเขียนโปรแกรมกิจกรรม ผู้เขียนขอนำมากล่าวถึงก่อน แต่จะอธิบายเฉพาะหน้าที่ (ส่วนเหตุผลจะอยู่ในหัวข้อ "งานเขียนโปรแกรมกิจกรรม") ได้แก่

- stndNum เพื่อเก็บจำนวนช่องของอะเรย์ตามที่ผู้เขียนโปรแกรมใช้งานจะกำหนดสำหรับการขยาย buff ต่อไป

- buffSize เพื่อเก็บจำนวนช่องของอะเรย์ในขณะนั้นเริ่มด้วยค่าเดียวกับ stndNum และเพิ่มมากขึ้นเมื่อมีการขยาย buff

- lastPstn เก็บช่องใน buff ซึ่งเป็นตำแหน่งข้อมูลสุดท้ายเริ่มด้วยค่าศูนย์

- errnum เพื่อเก็บรหัสแสดงถึงข้อผิดพลาด

- break ในการทำงานเป็นวงรอบที่ซ้อนกันอยู่ บางครั้งเราต้องการให้หลุดออกไปวงรอบนอกสุดหรือที่ต้องการ จึงได้กำหนดตัวแปรที่เป็นแฟล็กให้เป็นจริงเมื่อต้องการหลุดออกจากวงรอบ (ส่วนมากเมื่อมีข้อผิดพลาดเกิดขึ้น)

- extra กำหนดขึ้นเพื่อให้หน่วยความจำลงตัวเป็น 2 ไบต์เท่านั้นเพื่อการใช้งานในอนาคต ความจริงไม่ถูกต้องนักแต่คิดว่าไม่เสียหายอะไร

กิจกรรมรอง

จากการที่ต้องขยาย buff ที่เต็มแล้วจึงต้องมีโปรแกรมย่อยสำหรับดำเนินการในเรื่องนี้ และควรเป็นโปรแกรมย่อยภายในสกุล (ไม่ถือเป็นกิจกรรม) แต่ในสกุลลูกอาจมีการกำหนดอะเรย์ทำงานขนานกับ buff ซึ่งจะต้องเพิ่ม (หรือลด) ขนาดตาม buff ดังนั้นจึงต้องให้โปรแกรมที่จะดำเนินการในเรื่องนี้เป็นกิจกรรม (จะเห็นว่าในการออกแบบควรมองไปข้างหน้าบ้าง โดยเฉพาะเมื่อต้องทำเป็นเวอร์ชวล) กิจกรรมเหล่านี้ดังแสดงในรูปที่ 2 ได้แก่

- ExpandBuff เพื่อการขยาย buff โดยอัตโนมัติเมื่อ buff เต็มโดย AddItem และ InsertItem เป็นผู้เรียกใช้ผ่าน BuffAvail (และไม่มีการเรียกใช้โดยตรงในสกุลนี้) ดังนั้นหากต้องการให้บัฟเฟอร์ในสกุลลูกหลานขยายตัวโดยอัตโนมัติจะต้องให้ ExpandBuff เป็นเวอร์ชวล แต่ในการออกแบบ เมื่อมี ExpandBuff ก็ควรมี CrushBuff เพื่อการลดขนาดบัฟเฟอร์ แต่ไม่เห็นควรให้ลดขนาดเมื่อถูกถอนข้อมูลคือ DelItem ดังนั้นการใช้ CrushBuff จึงต้องเรียกใช้โดยตรง ทำให้เป็นเวอร์ชวลไม่ได้ (และตัวของมันเองก็ไม่จำเป็นต้องทำเป็นเวอร์ชวล) เพื่อให้เหมือนกัน เพราะเป็นกิจกรรมในแบบเดียวกัน จึงสามารถเรียกใช้ ExpandBuff ได้ด้วย นั่นคือต้องให้กิจกรรมทั้งสองนี้เรียกใช้กิจกรรมที่เป็นเวอร์ชวล ดังจะกล่าวต่อไป

```
      function   ExpandBuff : Boolean;
      function   DoExpandBuff : Boolean; VIRTUAL;
      function   CrushBuff : Boolean;
      function   DoCrushBuff : Boolean; VIRTUAL;
      function   BuffBuff(num : Word; size : Byte) : Boolean;
      procedure  FreeItem(pstn : Word); VIRTUAL;

      function   HeapAvail(bytereq : Word) : Boolean;
      procedure  SetErrorBreak(num : Word);
      DESTRUCTOR  Fin;  VIRTUAL;

(*  Internal sub-routine   *)
      function   BuffAvail : Boolean;
      function   NotInBuff(pstn : Word) : Boolean;
      function   DiffNum : Word;
                   END;
```

รูปที่ 2

- DoExpand Buff คือส่วนหนึ่งของ ExpandBuff ที่แยกมาทำเป็นเวอร์ชวล

- CrushBuff เพื่อการลดขนาดของ buff

- DoCrushBuff คือส่วนของ CrushBuff ที่แยกมาทำเป็นเวอร์ชวล

กิจกรรมประกอบ

ในขั้นตอนของการเขียนโปรแกรมกิจกรรมจะพบว่าต้องสร้างโปรแกรมย่อยเพิ่มขึ้น ซึ่งจะต้องพิจารณาว่าจะให้เป็นโปรแกรมย่อยภายในสกุลหรือกำหนดให้เป็นกิจกรรม การกำหนดให้เป็นกิจกรรมมีดังนี้คือ

- BuffBuff (ไม่ทราบว่าจะให้ชื่ออะไรดี) เป็นส่วนหนึ่งของ DoExpandBuff กับ DoCrushBuff ที่ซ้ำกัน ทำหน้าที่กำหนดอะเรย์ใหม่ในฮีพ ถ่ายข้อมูลจากอะเรย์เดิมมาอะเรย์ใหม่ และยกเลิกอะเรย์เดิม สามารถนำไปใช้ในสกุลลูกหลานได้จึงให้เป็นกิจกรรม

- FreeItem เมื่อให้ DelItem หรือ ReplaceItem อาจต้องยกเลิกข้อมูลเดิมในฮีพ จึงมอบให้ FreeItem ดำเนินการ จากแนวความคิดที่ว่าในสกุลลูกหลานอาจมีการกำหนดอะเรย์และเก็บข้อมูลในฮีพ ซึ่งจะต้องถูกยกเลิกโดยอัตโนมัติด้วย จึงต้องให้ FreeItem เป็นกิจกรรมและเวอร์ชวล

- HeapAvail เพื่อตรวจว่าที่ว่างในฮีพเพียงพอแก่ความต้องการหรือไม่ ซึ่งต้องใช้ในสกุลลูกหลานแน่จึงทำให้เป็นกิจกรรม

- SetErrBreak เมื่อเซต errnum ควรจะเซต break ด้วย จึงทำเป็นโปรแกรมย่อยไว้และจัดเป็นกิจกรรม

โปรแกรมย่อยภายใน

โปรแกรมย่อยที่ใช้เฉพาะภายในสกุล Collection เพื่อสนับสนุนการทำงานของโปรแกรมกิจกรรมดังกล่าว ได้แก่

- BuffAvail เพื่อตรวจว่าได้เก็บข้อมูลของ buff จนเต็มหรือยัง ถ้าเต็มจะขยาย buff ให้

- NotInBuff เพื่อตรวจว่าตำบลที่อ้างถึงอยู่ใน buff หรือไม่

- DiffNum ให้ค่า 2.5% ของ stndNum

การระบุสกุล

ขั้นตอนต่อไปของการเขียนโปรแกรมในแบบโอโอพีคือ นำตัวแปร กิจกรรมและโปรแกรมย่อยที่ระบุไว้ดังกล่าวไประบุสกุลดังแสดงในรูปที่ 1 และ 2 (ในการป้อนให้กระทำต่อเนื่องกัน) คำว่าระบุสกุลเดิมผู้เขียนใช้คำว่ากำหนดสกุล แต่เพื่อให้ตรงกับ C++ จะใช้คำว่าการกำหนดสกุลเป็นคำรวมของคำว่าการระบุ (declaration) กับการนิยาม (implement) การกำหนดสกุลไม่มีคำในภาษาอังกฤษ

```
CONST   NoErr         = 0;     { No Error }
        MemFull       = 1;     { Memmory full }
        FiNotFound    = 2;     { File not found }
        PathNotFound  = 3;     { Path not found }
        TooManyFi     = 4;     { Too many full }
        FiDenied      = 5;     { File denied }
        OldFi         = 6;     { Old file }

        DiskFull      = 102;   { Disk full }

        VitalErr      = 500;   { Vital error }

        Esc           = #27;
        colsize       = 6;

VAR   oldBuff,newBuff : Pointer;
      oldnum,newnum : Word;
```

รูปที่ 3

นอกแบบแผน

ในการเขียนโปรแกรมนี้ผู้เขียนพยายามที่จะเขียนตามแบบของโอโอพีและปาสคาล แต่ก็อดไม่ได้ที่จะเขียนออกไปนอกแบบซึ่งมีทั้งมากและน้อย ตรงตามที่หนังสือบางเล่มได้กล่าวไว้ว่าเป็นการเขียนโปรแกรมของผู้ที่ติดยึดการเขียนโปรแกรมแบบเก่า แต่ผู้เขียนคิดว่ากฎทุกกฎมีข้อยกเว้น จึงขอนำมาเรียนให้ทราบตามที่รวมไว้ในรูปที่ 3 กล่าวคือ

ค่าคงตัว

ได้กำหนดค่าคงตัวไว้เพื่อใช้ในโปรแกรม จะได้ไม่ต้องเขียนเป็นตัวเลข ทำให้อ่านโปรแกรมได้ง่ายกว่ากัน เรื่องนี้ไม่ผิดไปจากวิธีแบบโอโอพีหรอกครับ ถ้าผู้เขียนจะกำหนดไว้เพียง 2 ตัวคือ NoErr กับ MemFull เพราะนำไปใช้ใน Collection แต่ค่าที่เหลือ (เว้น Esc กับ calsize) เป็นการกำหนดเพื่ออนาคต ซึ่งในโอโอพีของ C++ มีข้อห้ามไว้ไม่ให้ทำ แต่ผู้เขียนเห็นว่าค่าคงตัวพวกนี้เป็น DOS error ซึ่งต้องใช้อยู่แล้ว และผู้เขียนต้องการเตือนใจผู้เขียนโปรแกรมคนต่อ ๆ ไปให้กำหนดค่า error ตั้งแต่ 501 เป็นต้นไป แต่ไม่ใช่เรื่องรุนแรงอะไรจึงกำหนดไว้ล่วงหน้า สำหรับค่า VitalErr เขียนไว้เพื่อเป็นที่สังเกตเท่านั้นไม่ได้นำไปใช้ อีกประเด็นหนึ่งที่ผิดหลักโอโอพีคือผู้เขียนได้กำหนดไว้นอกสกุล ก็เพราะไม่มีทางเลือก (ใน C++ ให้กำหนดค่าคงตัวในสกุลได้)

ตัวแปร

ได้กำหนดตัวแปรเป็นโคบอลเพื่อใช้เป็นส่วนกลางคือ oldBuff, newBuff, oldnum และ newnum เป็นตัวแปรที่จะเก็บค่าบัฟเฟอร์และความยาวไว้ชั่วคราวในขณะที่ขยายหรือลดขนาดของ buff (และที่ทำหน้าที่เดียวกันนี้ในสกุลลูกหลานต่อไป) การกำหนดตัวแปรเหล่านี้ผิดทั้งหลักการในการเขียนโปรแกรมในแบบโอโอพี (อย่างมาก) ในการเขียนโปรแกรมภาษาปาสคาลโดยทั่วไป (น้อยหน่อย) ที่ถูกควรนำไปกำหนดไว้ในสกุล Collection แต่ผู้เขียนเห็นว่าเกะกะ เปลืองหน่วยความจำ เพราะไม่ว่าเราจะกำหนดนิมิตของ Collection เป็นจำนวนเท่าไร ก็ไม่จำเป็นต้องมีตัวแปรเหล่านี้เป็นของตนเอง ไม่มีทางที่จะใช้ตัวแปรเหล่านี้พร้อมกัน แต่อาจผลัดกันใช้ได้ และคงเป็นเพราะผู้เขียนยังยึดถือการเขียนโปรแกรมในแบบเก่าจึงกำหนดไว้เช่นนี้

งานเขียนโปรแกรมกิจกรรม

ขั้นตอนสุดท้ายของการเขียนโปรแกรมในแบบโอโอพีคือ การเขียนโปรแกรมกิจกรรม ซึ่งใน C++ เรียกว่าเป็นการนิยามสกุลดำเนินการเช่นเดียวกับการเขียนโปรแกรมโดยทั่วไป โดยนำกิจกรรมที่ระบุไว้แล้วมาเขียน ได้แก่

AddItem

นำข้อมูลไปเก็บใน buff ครั้งละ 1 ข้อมูลแบบซีเควนเชียล (sequential) คือจัดเก็บข้อมูลเรียงลำดับกันไปจากต้นอะเรย์ (ช่องที่ 1) โดยนำข้อมูลใหม่ไปเก็บต่อท้ายข้อมูลเดิมซึ่งจะทำได้โดย

- จะต้องมีตัวแปรทำหน้าที่เป็นอินเด็กซ์ชี้ข้อมูลสุดท้ายใน buff คือ lastPstn (ตามที่กำหนดตัวแปรไว้แล้ว) ให้ค่าเริ่มต้นเป็นศูนย์
- เมื่อจะเก็บข้อมูลให้เพิ่มหนึ่ง

lastPstn แล้วนำข้อมูลไปเก็บ ณ ช่องที่ lastPstn ชี้อยู่ เริ่มต้นตั้งแต่ช่องที่ 1 เป็นต้นไป

– ก่อนเก็บจะต้องตรวจสอบก่อนว่า ที่เก็บข้อมูลเต็มหรือไม่ หากเต็มแล้วจะต้องขยาย หรือถ้าไม่ได้จะต้องแจ้งให้ทราบ โดยมอบให้ BuffAvail รับไปดำเนินการรวมทั้งให้เพิ่มหนึ่ง lastPstn ด้วย

ในขั้นตอนของการเขียนโปรแกรมย่อยจะต้องพิจารณาด้วยว่าได้กำหนดให้เป็นฟังก์ชันหรือโพรซีเดอร์ เนื่องจากต้องแจ้งให้ทราบว่าเก็บได้หรือไม่ จึงควรเป็นฟังก์ชัน ซึ่งจะต้องพิจารณาต่อไปว่าควรให้เป็นฟังก์ชันแบบใด ถ้าเป็นแบบบูลีนก็จะทราบว่าเก็บได้หรือไม่เท่านั้น จึงให้เป็นฟังก์ชันแบบ Word ซึ่งจะให้ค่าคืนมาเป็นช่องในอะเรย์ที่นำข้อมูลไปเก็บ เพื่อผู้เขียนโปรแกรมคนต่อไปสามารถนำไปใช้ หากเก็บไม่ได้ (หน่วยความจำเต็ม) จะให้ค่าคืนกลับมาเป็นศูนย์ เขียนเป็นโปรแกรมได้ดังแสดงในรูปที่ 4

ต่อไปผู้เขียนของคกล่าวถึงเรื่องที่จะให้เป็นโปรแกรมย่อยแบบใด ขอให้ลองเปรียบเทียบดูก็แล้วกัน สำหรับกิจกรรมบางกิจกรรมที่ใกล้เคียงกันให้พิจารณาจากโปรแกรมที่ให้มา และเป็นเรื่องของการเขียนโปรแกรมโดยทั่วไป ไม่ว่าจะเป็นโอโอพีหรือไม่

DelItem

ยกเลิกข้อมูล ณ ช่องที่ pstn การยกเลิกข้อมูลคือการถอนข้อมูลจาก buff และยกเลิกข้อมูลในฮีพตามโปรแกรมในรูปที่ 5 โดย

– ตรวจว่ามีข้อมูล ณ ตำแหน่งนี้ใน buff หรือไม่ ถ้าไม่มีจะไม่ดำเนินการต่อ หากมีจะดำเนินการต่อไป

– ยกเลิกข้อมูลในฮีพ โดยมอบให้ FreeItem รับไปดำเนินการ

– การถอนข้อมูลจาก buff ดำเนินการโดยเลื่อนข้อมูลหลัง Pstn มาข้างหน้า 1 ตำแหน่ง และลดหนึ่งค่า lastPstn

ReplaceItem

นำข้อมูลใหม่ไปแทนที่ข้อมูลเดิม ณ ตำแหน่งที่ pstn ตามรูปที่ 5 โดย

– ตรวจว่ามีข้อมูลเก่าใน buff หรือไม่

```
Implementation

PROCEDURE  Collection.FreeItem(pstn : Word);
BEGIN
  if  buff^[pstn].size <> 0  then
    FreeMem(buff^[pstn].dptr, buff^[pstn].size);
  buff^[pstn].size := 0;  buff^[pstn].dptr := nil;
END;

FUNCTION  Collection.BuffAvail : Boolean;
BEGIN
  inc(lastPstn);
  if  lastPstn < buffSize  then  BuffAvail := True
  else  BuffAvail := ExpandBuff;
END;

FUNCTION   Collection.HeapAvail(bytereq : Word) : Boolean;
BEGIN
  if  MaxAvail > bytereq  then  HeapAvail := True
  else  begin  HeapAvail := False;
               SetErrorBreak(memfull)  end;
END;

PROCEDURE  Collection.SetErrorBreak(num : Word);
BEGIN   errnum := num;  break := True  END;

FUNCTION  Collection.NotInBuff(pstn : Word) : Boolean;
BEGIN   NotInBuff := (pstn < 1) or (pstn > lastPstn)  END;
```

รูปที่ 4

```
CONSTRUCTOR  Collection.Init(numOfItem : Word);
VAR  idx : Word;
BEGIN
  stndNum := numOfItem;  buffSize := numOfItem;  errnum := 0;
  if  HeapAvail(colsize*buffSize)  then
    GetMem(buff,colsize*buffSize)
  else  begin  SetErrorBreak(MemFull);  Exit  end;
  for  idx := 1 to buffSize  do
    begin  buff^[idx].size := 0;  buff^[idx].dptr := nil  end;
  lastPstn := 0;  break := False;
END;

FUNCTION  Collection.AddItem(len : Word; newitem : Pointer) : Word;
BEGIN
  if  not BuffAvail  then  begin  AddItem := 0;  Exit  end;
  AddItem := lastPstn;
  buff^[lastPstn].size := len;
  buff^[lastPstn].dptr := newitem;
END;
```

รูปที่ 5

- หากมีให้ยกเลิกข้อมูลเดิมในฮีพแล้วนำข้อมูลใหม่ไปแทนที่

InsertItem

แทรกข้อมูล ณ ตำแหน่ง pstn ตามรูปที่ 6 โดยตรวจว่าตำแหน่งนี้ pstn อยู่ใน buff หรือไม่ และ buff ว่างหรือไม่ ถ้าอยู่และว่างจะแทรกข้อมูลโดย

- เลื่อนข้อมูล ณ ตำแหน่ง pstn เป็นต้นไปถอยหลัง 1 ตำแหน่ง (เป็นการเพิ่มหนึ่งค่า lastPstn ด้วย)
- นำข้อมูลมาเก็บ ณ ตำแหน่ง pstn

GetItem

ให้ค่าข้อมูล ณ ตำแหน่ง Pstn ซึ่งก็คือการอ่านข้อมูลนั่นเอง ดังโปรแกรมในรูปที่ 6 สังเกตว่า GetItem เป็นฟังก์ชันแบบ Pointer ซึ่งจะให้ค่าแก่พอยน์เตอร์แบบใดก็ได้ และโปรแกรมที่อ่านจะต้องทราบว่าตนอ่านข้อมูลแบบใด

BuffBuff

ตามรูปที่ 7 มีหน้าที่ขยายหรือลดขนาดบัฟเฟอร์ใดก็ได้ที่ให้ oldBuff ชี้อยู่ (เพื่อใช้ในสกุลลูกด้วย) โดย

- กำหนดหน่วยความจำใหม่ในฮีพใหม่ มีขนาดเท่ากับ num (จำนวนช่องของอะเรย์) คูณด้วย size (ขนาดของช่องเป็นไบต์)
- ให้ newBuff ชี้ที่บัฟเฟอร์ใหม่นี้
- ลอกข้อมูลจากบัฟเฟอร์เดิมมาลงบัฟเฟอร์ใหม่ แล้วยกเลิกหน่วยความจำในฮีพที่เป็นบัฟเฟอร์เดิม

ExpandBuff

ขยายขนาดของบัฟเฟอร์ buff เดิมให้เพิ่มขึ้น 25 เปอร์เซ็นต์ ตามรูปที่ 8 โดย

- ให้ขนาด (จำนวนช่อง) ของอะเรย์ที่เป็นบัฟเฟอร์เดิมแก่ oldnum และขนาดของอะเรย์ใหม่แก่ newnum

```
FUNCTION   Collection.DelItem(pstn : Word) : Boolean;
VAR  idx : Word;
BEGIN
  if  NotInBuff(pstn)  then  begin  DelItem := False;  Exit  end;
  DelItem := True;  FreeItem(pstn);
  for  idx := pstn to lastPstn-1  do  buff^[idx] := buff^[idx+1];
  buff^[lastPstn].dptr := nil;  dec(lastPstn);
END;


FUNCTION   Collection.ReplaceItem(pstn,len : Word;
                                  newitem : Pointer) : Boolean;
BEGIN
  if  NotInBuff(pstn)  then
    begin  ReplaceItem := False;  Exit  end;
  ReplaceItem := True;  FreeItem(pstn);
  buff^[pstn].size := len;
  buff^[pstn].dptr := newItem;
END;
```

รูปที่ 6

```
FUNCTION  Collection.InsertItem(pstn,len : Word;
                                newitem : Pointer) : Boolean;
VAR   idx : Word;
BEGIN
  if  NotInBuff(pstn) or (not BuffAvail)  then
    begin  InsertItem := False;  Exit  end;
  InsertItem := True;
  for  idx := lastPstn downto pstn+1  do
    buff^[idx] := buff^[idx-1];
  buff^[pstn].size := len;
  buff^[pstn].dptr := newItem;
END;


FUNCTION   Collection.GetItem(pstn : Word) : Pointer;
BEGIN
  if  NotInBuff(pstn)  then  begin  GetItem := nil;  Exit  end;
  GetItem := buff^[pstn].dptr
END;
```

รูปที่ 7

```
FUNCTION  Collection.BuffBuff(num : Word; size : Byte) : Boolean;
VAR   len,oldlen,newlen : Word;
BEGIN
  len := size * num;  oldlen := size * oldnum;
  newlen := size * newnum;  BuffBuff := True;
  if  HeapAvail(newlen)  then  GetMem(newBuff,newlen)
  else  begin  BuffBuff := false;  Exit  end;
  Move(oldBuff^,newBuff^,len);
  FreeMem(oldBuff,oldlen);
END;
```

รูปที่ 8

ซึ่งค่าทั้งสองนี้โปรแกรมลูกหลานจะนำไปใช้ด้วย

- การหาขนาดของอะเรย์ใหม่มอบให้ DiffNum (รูปที่ 7) เป็นผู้ดำเนินการโดยหาค่าที่เพิ่ม 25% แต่ถ้าน้อยกว่า 1 ให้เป็น 1
- มอบให้ DoExpandBuff ดำเนินการในส่วนที่เหลือ

DoExpandBuff

ดำเนินการขยายขนาดของบัฟเฟอร์ในส่วนที่เกี่ยวข้องกับบัฟเฟอร์แต่ละตัว (จึงทำเป็นเวอร์ชวล) ตามรูปที่ 8 โดย

- ให้ oldBuff ชี้ไปที่บัฟเฟอร์เดิมของตน
- ให้ BuffBuff ขยายบัฟเฟอร์ให้ตามขนาดที่ต้องการ
- หากขยายได้จะให้ช่องที่ขยายใหม่เป็นช่องว่าง (ไม่มีข้อมูล)

CrushBuff/DoCrushBuff

ลดขนาดของบัฟเฟอร์ลง 25% มีวิธีการดำเนินการเช่นเดียวกับ Expand-Buff ตามรูปที่ 9

Fin

ตามรูปที่ 10 เป็น destructor เพื่อการยกเลิกนิมิตของสกุลนี้ เช่น เพื่อนำหน่วยความจำไปใช้ในงานอื่น หรือเพื่อการเคลียร์บัฟเฟอร์ (ซึ่งเมื่อ Fin แล้วต้อง Init ใหม่) โดย

- ยกเลิกข้อมูลตัวจริงในฮีพ ขอให้สังเกตว่าไม่ได้เรียกใช้ FreeItem เพราะเป็นเวอร์ชวลอาจทำให้สับสนได้
- ยกเลิกบัฟเฟอร์ buff
- เมื่อเรียกใช้โปรแกรมย่อยนี้แล้ว เทอร์โบปาสคาลจะยกเลิกตาราง VMT (Virtual Method Table) ซึ่งสร้างเมื่อเรียกใช้ Init ทำให้การยกเลิกหน่วยความจำที่กำหนดในฮีพทำได้หมดจด

สำหรับกิจกรรมและโปรแกรมย่อยอื่นที่ไม่ต้องการคำอธิบาย ผู้เขียนได้รวบรวมไว้ในรูปที่ 11

โปรแกรมทดสอบ

การป้อนโปรแกรมดังกล่าวข้างต้นนั้นขอให้ป้อนเรียงลำดับจากรูปที่ 1 ถึง 11 แล้วคอมพายล์เป็นไฟล์ .TPU เก็บไว้ และเพื่อทดสอบการทำงานผู้เขียนได้

```
FUNCTION  Collection.DiffNum : Word;
BEGIN
  if  stndNum <= 4  then  DiffNum := 1
  else  DiffNum := stndNum div 4;
END;

FUNCTION   Collection.ExpandBuff : Boolean;
BEGIN
  oldnum := buffSize;  newnum := buffSize + DiffNum;
  ExpandBuff := DoExpandBuff
END;

FUNCTION  Collection.DoExpandBuff : Boolean;
VAR   idx : Word;   ok : Boolean;
BEGIN
  oldBuff := buff;  ok := BuffBuff(oldNum,colsize);
  DoExpandBuff := ok;
  if  not ok  then  Exit;
  buff := newbuff;  buffSize := newnum;
  for  idx := oldnum+1 to buffSize  do
    begin  buff^[idx].size := 0;  buff^[idx].dptr := nil  end;
END;
```

รูปที่ 9

```
FUNCTION   Collection.CrushBuff : Boolean;
BEGIN
  CrushBuff := False;
  if  buffSize = stndNum  then  Exit;
  oldnum := buffSize;  newnum := buffSize - Diffnum;
  if  newnum < stndNum  then  newnum := stndNum;
  CrushBuff := DoCrushBuff;
END;

FUNCTION  Collection.DoCrushBuff : Boolean;
VAR    ok : Boolean;
BEGIN
  oldBuff := buff;  ok := BuffBuff(newnum,colsize);
  DoCrushBuff := ok;
  if  not ok  then  Exit;
  buff := newbuff;  buffSize := newnum;
END;
```

รูปที่ 10

```
DESTRUCTOR  Collection.Fin;
VAR  idx : Word;
BEGIN
  for  idx := 1 to lastPstn  do
    if  buff^[idx].size <> 0  then
      FreeMem(buff^[idx].dptr, buff^[idx].size);
  FreeMem(buff,colsize*buffSize);
END;


END.
```

รูปที่ 11

```
PROGRAM TestCollection;

USES collect;

TYPE Str19 = String[19]; Ptr2StrT = ^Str19;
     Ptr2IntT = ^Integer;
VAR datclct : Collection;
    sttmem : LongInt;
    idx,res,strlen,num1,num2,num3 : Integer;
    ok : Boolean;
    str1,str2,str3 : Str19;
    hstr,strx : Ptr2StrT; tem : Pointer;


procedure WriteStr(sptr : Ptr2StrT);
begin WriteLn(sptr^) end;


procedure WriteInt(iptr : Ptr2IntT);
begin Write(iptr^,' ') end;


function Remain : Integer;
begin remain := datclct.buffsize - datclct.lastPstn end;


procedure DispItem;
var id : Integer;
begin
 WITH datclct DO
  BEGIN
  for id := 1 to 3 do
    begin Write(id,'='); WriteInt(buff^[id].dat) end;
  WriteLn;
  for id := 4 to lastPstn do
    begin Write(id,' = '); WriteStr(buff^[id].dat) end;
  WriteLn('buff and remain = ',BuffSize,' ',Remain);
  WriteLn('Mem Avail = ',MemAvail);
  END;
 Write('..... Press ENTER ..... '); ReadLn;
end;
```

รูปที่ 12

```
BEGIN { MAIN }
  sttmem := MemAvail; strlen := SizeOf(Str19);
  WriteLn('At start mem avail = ',sttmem,' .....');
  GetMem(hstr,strlen); GetMem(strx,strlen);
  datclct.Init(3);

  num1 := 1; num2 := 2; num3 := 3;
  str1 := 'First string'; str2 := 'Second string';
  str3 := 'Third string';
  hstr^ := 'Heap string'; strx^ := 'Replace string';

  WITH datclct DO
    BEGIN

    WriteLn('Before add item ',BuffSize,' ',Remain);
    res := AddItem(0,@num1);   res := AddItem(0,@num2);
    res := AddItem(0,@num3);
    res := AddItem(0,@str1);   res := AddItem(0,@str2);
    res := AddItem(0,@str3);   res := AddItem(strlen,hstr);
    DispItem;


    WriteLn('..... Delete Item 4 .....');
    if DelItem(4) then DispItem;

    if CrushBuff then WriteLn('Can crush ',BuffSize,' ',Remain)
    else WriteLn('Can not crush'); ReadLn;

    WriteLn('..... Replace Item 5 .....');
    if ReplaceItem(5,strlen,strx) then DispItem;

    WriteLn('..... Insert Item 4 .....');
    str1 := 'Insert string';
    if InsertItem(4,0,@str1) then DispItem;

    END;

    WriteLn('Finish');  datclct.Fin;
    WriteLn('.At start mem avail = ',sttmem,' .....');
    WriteLn('..... Mem Avail = ',MemAvail,' .....');

Write('END OF PROGRAM press ENTER'); ReadLn;
END.
```

รูปที่ 13

เขียนโปรแกรมทดสอบไว้ในรูปที่ 12 และ 13 โปรแกรมนี้แสดงถึงความสามารถของ Collection ในการเก็บข้อมูลต่างแบบกันได้ โดยให้เก็บข้อมูลทั้งแบบเลขจำนวนและสตริง ทั้งชนิดสแตติก (ตัวแปร) และไดนามิก (เก็บในฮีพ) และได้แสดงการถอนข้อมูล (delete) สับเปลี่ยน (replace) และการแทรก (insert) ซึ่งเป็นสื่อความหลักของ Collection

ตอนต้นโปรแกรมก่อนที่จะดำเนินการใดได้กำหนดให้เก็บจำนวนไบต์ที่ว่างในฮีพไว้ในตัวแปร sttmem เพื่อนำมาแสดงเปรียบเทียบในตอนท้ายของโปรแกรมว่าได้ยกเลิกหน่วยความจำในฮีพได้อย่างหมดจด

หนทางข้างหน้า

ผู้เขียนหวังว่าท่านผู้อ่านคงทำความเข้าใจเรื่องการเขียนโปรแกรมในแบบโอโอพีได้พอสมควร ตอนต่อไปผู้เขียนจะนำสกุล Collection ไปใช้กับซอฟต์แวร์ชิพที่ค้างอยู่ และเป็นโอกาสชี้ให้เห็นถึงข้อดีข้อเสียของการเขียนโปรแกรมในแบบโอโอพีด้วยครับ